METHOD AND SYSTEM FOR MULTI-PROCESSOR FFT/IFFT WITH MINIMUM INTER-PROCESSOR DATA COMMUNICATION

PRIORITY CLAIM

[1] This application claims priority from Indian patent application No. 127/Del/2003, filed February 17, 2003, which is incorporated herein by reference.

TECHNICAL FIELD

[2] The present invention relates to the field of digital signal processing. More particularly the invention relates to a device and method for providing an FFT/IFFT implementation providing minimum inter-processor communication overhead and less silicon area in a multiprocessor system.

BACKGROUND

[3] The class of Fourier transforms that refer to signals that are discrete and periodic in nature are known as Discrete Fourier Transforms (DFT). The discrete Fourier transform (DFT) plays a key role in digital signal processing in areas such as spectral analysis, frequency domain filtering and polyphase transformations.

[4] The DFT of a sequence of length N can be decomposed into successively smaller DFTs. The manner in which this principle is implemented falls into two classes. The first class is called a "decimation in time" approach and the second is called a "decimation in frequency" method. The first derives its name from the fact that in the process of arranging the computation into smaller transformations the sequence "x(n)" (the index 'n' is often associated with time) is decomposed into successively smaller subsequences. In the second general class the sequence of DFT coefficients "x(k)" is decomposed into smaller subsequences (k denoting frequency). The present invention employs "decimation in time".

[5] Since the amount of storing and processing of data in numerical computation algorithms is proportional to the number of arithmetic operations, it is generally

accepted that a meaningful measure of complexity, or of the time required to implement a computational algorithm, is the number of multiplications and additions required. The direct computation of the DFT requires "$4N^2$" real multiplications and "N(4N-2)" real additions. Since the amount of computation and thus the computation time is approximately proportional to "N^2" it is evident that the number of arithmetic operations required to compute the DFT by the direct method becomes very large for large values of "N". For this reason, computational procedures that reduce the number of multiplications and additions are of considerable interest. The Fast Fourier Transform (FFT) is an efficient algorithm for computing the DFT.

[6] The conventional method of implementing an FFT or Inverse Fourier Transform (IFFT) uses a radix-2 / radix-4 / mixed-radix approach with either "decimation in time (DIT)" or a "decimation in frequency (DIF)" approach.

[7] The basic computational block is called a "butterfly" ---- a name derived from the appearance of flow of the computations involved in it. **FIG. 1** shows a typical radix-2 butterfly computation. 1.1 represents the 2 inputs (referred to as the "odd" and 'even" inputs) of the butterfly and 1.2 refers to the 2 outputs. One of the inputs (in this case the odd input) is multiplied by a complex quantity called the twiddle factor (W_N^k). The general equations describing the relationship between inputs and outputs are as follows:

$$X[k] = x[n] + x[n+N/2]W_N^k$$
$$X[k+N/2] = x[n] - x[n+N/2]W_N^k$$

[8] An FFT butterfly calculation is implemented by a z-point data operation wherein "z" is referred to as the "radix". An "N" point FFT employs "N/z" butterfly units per stage (block) for "$\log_z N$" stages. The result of one butterfly stage is applied as an input to one or more subsequent butterfly stages.

[9] Computational complexity for an N-point FFT calculation using the radix-2 approach = $O(N/2 * \log_2 N)$ where "N" is the length of the transform. There are exactly "$N/2 * \log_2 N$" butterfly computations, each comprising 3

complex loads, 1 complex multiply, 2 complex adds and 2 complex stores. A full radix-4 implementation on the other hand requires several complex load/store operations. Since only 1 store operation and 1 load operation are allowed per bundle of a typical VLIW processor that is normally used for such implementations, cycles are wasted in doing only load/store operations, thus reducing ILP (Instruction Level parallelism). The conventional nested loop approach requires a high looping overhead on the processor. It also makes application of standard optimization methods difficult. Due to the nature of the data dependencies of the conventional FFT/IFFT implementations, multi cluster processor configurations do not provide much benefit in terms of computational cycles. While the complex calculations are reduced in number, the time taken on a normal processor can still be quite large. It is therefore necessary in many applications requiring high-speed or real-time response to resort to multiprocessing in order to reduce the overall computation time. For efficient operation, it is desirable to have the computation as linearly scalable as possible --- in other words the computation time reducing in inverse proportion to the number of processors in the multiprocessing solution. Current multiprocessing implementations of FFT/IFFT however, do not provide such a linear scalability.

[10] US patent 6,366,936 describes a multiprocessor approach for efficient FFT. The approach defined is a pipelined process wherein each processor is dependent on the output of the preceding processor in order to perform its share of work. The increase in throughput does not scale proportionately to the number of processors employed in the operation.

[11] US patent 5,293,330 describes a pipelined processor for mixed size FFT. Here too, the approach does not provide proportional scalability in throughput, as it is pipelined.

[12] A scheme for parallel FFT/IFFT as described in "Parallel 1-D FFT Implementation with TMS320C4x DSPs" by the semiconductor group-Texas Instruments, uses butterflies that are distributed between two processors. In this implementation, inter processor communication is required because subsequent

computations on one processor depend on intermediate results from other processors. Every processor computes a butterfly operation on each of the butterfly pairs allocated to it and then sends half of its computed result to the processor that needs it for the next computation step and then waits for the information of the same length from another node to arrive before continuing computations. This interdependence of processors for a single butterfly computation does not support proportionate increase in output with increase in the number of processors.

[13] Our co-pending application no. 1208/D/02 describes a linearly scalable FFT/IFFT system. The system incorporates a shared memory wherein each processor accesses correct data samples from the shared memory. Distribution is such that no inter-processor communication is required during the butterfly computation. However there is a requirement of inter-processor communication between stages.

[14] Though a shared memory system is easier it is not very economical. This is because this approach requires multi port memories that are very expensive. Therefore a distributed memory system is more economical. The distributed memory architecture requires a media to communicate data among the processors. Hence it is desirable that the data communication among the processors in distributed memory architecture is minimum. Since the input data is distributed in equal size segments to each processor and each processor performs computations only on the data in its local memory, the memory requirement for individual processor reduces resulting in a lower requirement for silicon area and cost.

SUMMARY

[15] An aspect of the present invention is to overcome the above drawbacks and provide a device and method for implementing FFT/IFFT with minimum communication overhead among processors in a multiprocessor system using distributed memory.

[16] According to an aspect of the invention, a scalable method for implementing FFT/IFFT computations in multiprocessor architectures that provides improved

throughput by eliminating the need for inter-processor communication after the computation of the first "$\log_2 P$" stages for an implementation using "P" processing elements, comprises the steps of:

computing each butterfly of the first "$\log_2 P$" stages on either a single processor or each of the "P" processors simultaneously,

distributing the computation of the butterflies in all the subsequent stages among the "P" processors such that each chain of cascaded butterflies consisting of those butterflies that have inputs and outputs connected together, are processed by the same processor.

[17] The distributing of the computation of the butterflies subsequent to the first "$\log_2 P$" butterflies is achieved by assigning operand addresses of each set of butterfly operands to each processor in such a manner that the butterfly is processed by the same processor that computed the connected butterfly of the previous stage in the same chain of butterflies.

[18] The desired assignment of operand addresses is achieved by deriving the address of the first operand in the operand pair corresponding to the "i^{th}" stage of the computation from the address of the corresponding operand in the previous stage by inserting a "0" in the "$(i+1)^{th}$" bit position of the address, while the address of the second operand is derived by inserting a "1" in the "$(i+1)^{th}$" bit position of the operand address.

[19] The above method further includes computing of twiddle factors for the butterfly computations at each processor by initializing a counter and then incrementing it by a value corresponding to the number of processors "P" and appending the result with a specified number of "0"s.

[20] Another aspect of the present invention provides a system for obtaining scalable implementation of FFT/IFFT computations in multiprocessor architectures that provides improved throughput by eliminating the need for inter-processor communication after the computation of the first "$\log_2 P$" stages for an implementation using "P" processing elements, comprising:

a means for computing each butterfly of the first "$\log_2 P$" stages on either a single processor or each of the "P" processors simultaneously,

an addressing means for distributing the computation of the butterflies in all the subsequent stages among the "P" processors such that each chain of cascaded butterflies consisting of those butterflies that have inputs and outputs connected together, are processed by the same processor.

[21] According to one aspect of the present invention, the addressing means comprises addresses generation means for deriving the operand addresses of the butterflies subsequent to the first "$\log_2 P$" butterflies in such a manner that the butterfly is processed by the same processor that computed the connected butterfly of the previous stage in the same chain of butterflies.

[22] According to one aspect of the present invention, the address generation means is a computing mechanism for deriving the address of the first operand in the operand pair corresponding to the "i^{th}" stage of the computation from the address of the corresponding operand in the previous stage by inserting a "0" in the "$(i+1)^{th}$" bit position of the address, and deriving the address of the second operand by inserting a "1" in the "$(i+1)^{th}$" bit position of the operand address.

[23] The above system further includes a computing mechanism for address generation of twiddle factors for each butterfly on the corresponding processor according to a further aspect of the present invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[24] The present invention will now be explained with reference to the accompanying drawings, which are given only by way of illustration and are not limiting for the present invention.

[25] **FIG. 1** shows the basic structure of the signal flow in a radix-2 butterfly computation for a discrete Fourier transform.

[26] **FIG. 2** shows a 2-processor implementation of butterflies for a 16-point FFT, in accordance with an embodiment of the present invention.

[27] **FIG. 3** shows a 4-processor implementation of butterflies for a 16-point FFT, in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

[28] The following discussion is presented to enable a person skilled in the art to make and use the invention. Various modifications to the embodiments will be readily apparent to those skilled in the art, and the generic principles herein may be applied to other embodiments and applications without departing from the spirit and scope of the present invention. Thus, the present invention is not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

[29] **FIG. 1** has already been described in the background of the invention.

[30] **FIG. 2** shows the implementation for a 16 point FFT in a 2-processor architecture using an embodiment of the present invention. Dark lines are computed in one processor, and light lines in the other. The computational blocks are represented by '0'. The left side of each computational block is its input (the time domain samples) while the right side is its output (transformed samples). The present invention uses a mixed radix approach with decimation in time. The first two stages of the radix-2 FFT/IFFT are computed as a single radix-4 stage. As these stages contain only load/stores and add/subtract operations there is no need for multiplication. This leads to reduced time for FFT/IFFT computation as compared to that with full radix-2 implementation. The next stages have been implemented as radix-2. The three main nested loops of conventional implementations have been fused into a single loop which iterates "$N/2*(\log_2 N-2))/(\text{number of processor})$" times. Each processor is used to compute one butterfly in one loop iteration. Since there is no data dependency between different butterflies in this algorithm, both during and between stages, the computational load can be equally divided among the different processors, leading to a nearly linear scalable system. There is no data dependency between stages and therefore each processor is able to perform the butterfly computations on the data assigned to it without communicating with the other processors.

[31] The mechanism for assigning the butterflies in this manner consists of generating the addresses of inputs such that each processor computes a complete sequence of cascaded butterflies. An N-bit counter, where "N" is the number of

stages is used to derive the addresses used for variables corresponding to each butterfly stage in the computation. Two inputs are generated for the two operands of the butterfly. Introducing '0' in a specified position of the counter generates the address for input 1. Introducing 1 in a specified position of the counter generates the address for input 2. For address generation of twiddle factors a separate counter with a specified number of bits is initialized on each processor. The counter value is then appended with a specified number of zeroes. The counter is incremented by a value corresponding to the number of processors and appended with a specified number of zeroes to get the twiddle factor address of the next butterfly stage.

[32] Distribution of data samples to the other processor can be after stage 1 at the earliest. But to save on unnecessary multiplications it can also be done after stage 2. No inter-processor communication is required once data is distributed. As a result, the red line outputs need to be collected by one of the processor at the end of the computation.

[33] **FIG. 3** shows a 4-processor implementation for the 16-point FFT using this invention. Different line colors represent computation in each of the 4 processors.

[34] In the present embodiments of the invention each processor comprises one or more ALUs (Arithmetic Logic unit), multiplier units, data cache, and load/store units. Each processor has an individual memory and the distribution of butterflies is such that there is no inter-processor communication required after the distribution of data. The distribution of data takes place after "$\log_2 P$" stages where "P" is the number of processors. Inter-processor communication takes place only before and after all the computations have been completed. The amount of data communication overhead does not increase with an increase in the number of processors.

[35] The embodiments of the invention, though described for distributed memory can be applied to shared memory systems also. Our co-pending application no. 1208/D/02 provides a linearly scalable system for shared memory systems only.

[36] It will be apparent to those with ordinary skill in the art that the foregoing is merely illustrative and is not intended to be exhaustive or limiting, having been presented by way of example only and that various modifications can be made within the scope of the above embodiments of the invention.

[37] Accordingly, this invention is not to be considered limited to the specific examples chosen for purposes of disclosure, but rather to cover all changes and modifications, which do not constitute departures from the permissible scope of the present invention. The invention is therefore not limited by the description contained herein or by the drawings, but only by the claims.

[38] The described embodiments of the present invention may be utilized in a variety of different types of integrated circuits including digital signal processors and communications circuits, and such circuits may be contained in a variety of different types of electronic systems such as communications and computer systems.